Implementation of the van Emde Boas Tree

With application to Dijkstra and comparision with respect to Fibonacci and Binomial Heaps.

By
Snehashis Pal (2018201072)
Anuj Bansal (2018201096)

Deliverables

A comprehensive study of the efficiency of Fibonacci Heap, Binomial Heap and van Emde Boas Trees when applied to calculating the single source shortest path (Dijkstra) in multiple graph of sufficient order. The comparison will primarily be focused on execution time. The output will be in the form of multiple tables and/or graphs and will be on datasets taken from a very generalized graph structure i.e. vertices and edges, with no notion of any prior application domain. This is done to ensure that the comparison is not biased towards any system specific implementation. Data-sets used will be random in nature and thus the output will represent results in a very generalized model of a system using graphs.

Project Delivery Plan

Phase 1. Study the implementation of Binomial and Fibonacci Heaps. In particular the theoretical amortized/worst case runtimes of various operations for each data structures. Compare with normal implementation of the binary heap data structures.

Phase 2. Study of Dijkstra algorithm and its generalized implementation on Binomial and Fibonacci Heaps. The worst case run time using each data-structures.

Phase 3. Implementation of Binomial Heaps and Fibonacci Heaps in C++. Implement a generalized structure to represent graphs which can be reused to test on different implementation.

Phase 4. Study van Emde Boas Trees, its theoretical complexity and pseudo code.

Phase 5. Implement van Emde Boas Trees in C++ code.

Phase 6. Generate test cases with focus on large, random and generalized data-sets of graphs.

Phase 7. Apply test cases on the three different data-structures and compare efficiency based on execution time.

Phase 8. Document the results in the form of tables and graphs.

Technologies

1. Primarily developed in the C++ language, compiled using GNU compiler collection.

2. Test cases generated in Python.

3. Emacs for documentation.

Resources Used

1. ***Introduction to Algorithms*** by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein

2. GeekforGeeks-https://www.geeksforgeeks.org/

Repository

Github Repository

Plan for testing

Since this is an academic comparision on the efficiency of various data-structures on Dijkstra, system level testing is unnecessary.

It is of utmost importance to maintain consistency in the testing environment, and not necessarily subjecting each data-structure to best case scenarios. All data-structures will be subject to the same set of randomly generated test cases.

All tests will be performed on an Intel i5-8250u with a pre-specified core frequency.

GCC optimization level 3 (-O3 flag) will be used to compile all code to eliminate coding inefficiencies.

The results of each test case will be averaged over a set of runs. This is to ensure outliers do not affect the final results. Test cases will also span the entire gamut of possible graph structures.

Finally we will analyze the results with the theoretical complexities calculated and try and find out reasons for deviation from the norm.

End User document

Since this project primarily focuses on comparing the efficiencies as a result of applying various data structures on a standard algorithm i.e. Dijkstra, the end user are much likely the developers themselves or some external observers. The results shown is a representative of the approximate runtimes expected in a system using the same or similar data structures scaled up/down depending on the how different the end user system is. As such the results can be used as proof of concept of the viability of the end user system which will be using such data structures and similar operations on them.

Since this project has been developed and tested in a single processor enviornment , possibilities are there that the results observed in single processor system will be different from those observed in a multiprocessor systems. Results may also deviate because of the differences in processor’s frequencies and compilers optimazation levels. Even so the testing is done as machine independent as possible and users should only expect deviation of constant factors.